Appendix A5: Software Design Description

The Software Design Description shall include a design showing the following:

1. The decomposition into software items

*See notes attached below for software decomposition information.*

1. The interrelationship between software items.

*The various libraries used and their interrelation is documented in the attached notes.*

1. Concept of execution

*The concept of execution is documented in the attached notes below. See finite state machines in the notes below.*

1. External interfaces, including Input/Output description.

*Communication to and from ICAROUS is via the MAVLink protocol. Inputs to ICAROUS are documented in the attached notes. ICAROUS outputs all the MAVLink messages the autopilot streams out via the COM output UDP port. Any MAVLink compatible ground station can be used to communicate with ICAROUS. Additional python modules available in the repository extend the MAVProxy ground station with capabilities to visualize/input geofence and traffic information.*

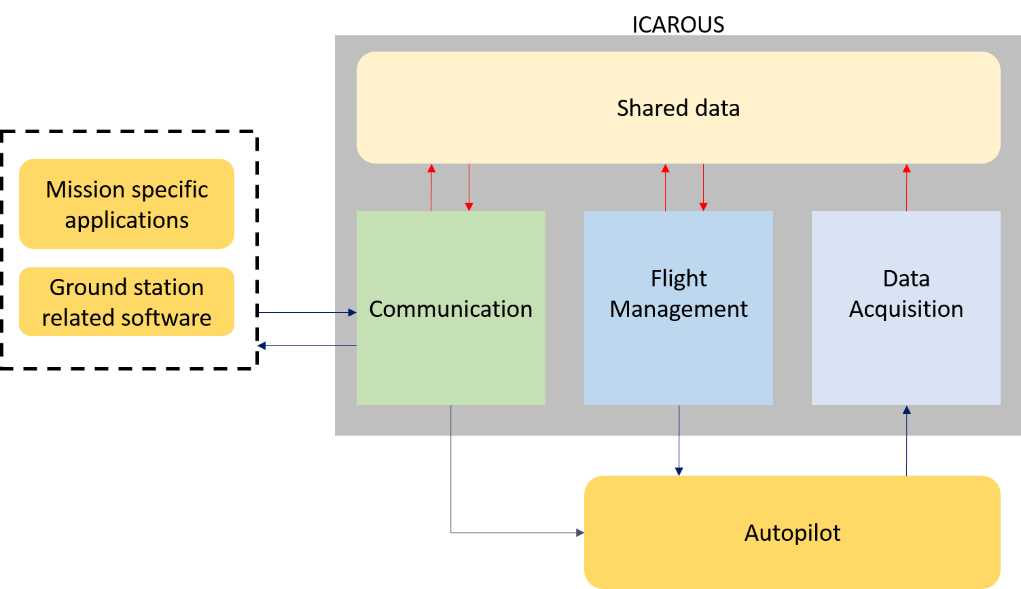
1. Traceability between the Software Requirements Specification and the Software Design Description.

* *Functions in the POLYCARP library are associated with the geofence requirement*
* *Functions in the ACCoRD library are associated with the traffic conflict resolution requirement.*

1. Include the software design decisions (e.g., assumptions, limitations, and reliability related items/concerns or constraints) and rationale for those decisions.

* *Relevant assumptions and software limitations will be updates as the research progresses.*

**ICAROUS software architecture**



At the top level there are three core threads within ICAROUS:

1. Data acquisition thread: The data acquisition thread (DAQ) is responsible for reading data from a specified input port (serial or socket). Data transfer is accomplished using MAVLink packets. The incoming MAVLink messages are stored in a shared data structure (class MAVLinkMessages) that can be accessed by all other threads.
2. Communication thread: Enables interaction with ICAROUS from a ground station via MAVLink messages. The COM thread passes MAVLink Messages streaming from the autopilot to the ground station. The COM thread also passes commands from the ground station (intended for the autopilot) directly to the autopilot.
3. Flight management thread: The flight management thread is responsible for making all flight related decisions, monitoring flight conditions for conflicts and resolving conflicts if necessary. Conflict detection and resolution is established using hierarchical finite state machines.

**ICAROUS package summary**

Below is a package summary of the Java implementation. The C++ implementation is very similar to the Java implementation. Below is a description of the various classes and interfaces that constitute the ICAROUS package.

class Interface

* Enables communication over a serial port or socket.
* Contains functions to read/write raw bytes and MAVLink packets.
* Provides a function to pass data between two interfaces.

class MAVLinkMessages

* A data structure to hold incoming MAVLinkMessages.
* Contains synchronized functions to decode incoming MAVLink packets.
* Functions to access to the latest MAVLink messages.

class AircraftData

* A data structure that holds relevant flight data such as aircraft state information, flight plan, geofence information and traffic information.
* Contains additional helper functions to read/send flight plans, geofences to/from a ground station/pixhawk flight controller.

class FlightManagementSystem

* Provides functions to send various commands to a pixhawk autopilot.
* Implements a finite state machine that enables autonomous flight through takeoff, climb, cruise and land.

class GeoFence

* A data structure that holds geofence related information.
* Contains member functions that make use of Polycarp routines to determine containment for keep in/out geofences and determine recovery points.

class GenericObject

* A data structure to hold traffic/obstacle/mission related object information.

class ConflictDetection

* A data structure that is used to capture conflict information.
* Provides function to detect various conflicts.
* The monitor function checks for geofence, traffic and standoff distance related conflicts.

class Resolution

* Provides functions to resolve conflict.
* The resolution function computes and executes a resolution in the form of a resolution plan or a maneuver sequence.

class Icarous

* The main entry point for ICAROUS. Sets up all the relevant threads and ICAROUS operating modes (active/passive/passthrough).
* In active mode, ICAROUS detects, logs and resolves conflicts.
* In passive mode, ICAROUS only detects and logs the conflicts. It does not interfere with the autopilot's operation.
* In passthrough mode, ICAROUS simply passes packets between a ground station and the autopilot.

class DAQ

* Implements the data acquisition thread.

class COM

* Implements the communication thread.

class BCAST

* Implements a broadcasting loop. This enables other onboard applications to obtained data from ICAROUS.

interface Mission

* Defines an interface that can be implemented by end users to perform mission specific operations.

**Libraries Used**

In addition to the above classes and interfaces, ICAROUS relies on formally verified algorithms to perform core functions such as detect and avoid other traffic and respect geofence constraints. These core functions can be found under the ACCoRD and Polycarp libraries respectively. To enable communication over serial ports, the Jssc library is utilized.

**Flightplan Input**

Flight plan inputs to ICAROUS must be consistent with the MAVLink waypoint protocol. A detailed description of the waypoint protocol can be found at [ http://qgroundcontrol.org/mavlink/waypoint\_protocol ](URL)

**Geofence Input**

ICAROUS expects geofence inputs according to the following protocol:

To begin geofence inputs, ICAROUS first expect a MAVLink command long message with the following parameters:

* command: MAV\_CMD\_DO\_FENCE\_ENABLE
* param1: 0 - on / 1 - off
* param2: geofence id
* param3: geofence type (0 - keepin/1 - keep out)
* param4: number of vertices
* param5: floor altitude (m)
* param6: ceiling altitude (m)

On receipt of the MAV\_CMD\_DO\_FENCE\_ENABLE command, ICAROUS sends out a FENCE\_FETCH\_POINT message requesting a vertex. Each geofence vertex must be encoded into the FENCE\_POINT message and sent in response to the requested vertex. After successfully receiving all the vertices, ICAROUS sends the MISSION\_ACK message.

**Traffic Input**

Traffic information can be send to ICAROUS via the MAVLink command long message with the following parameters:

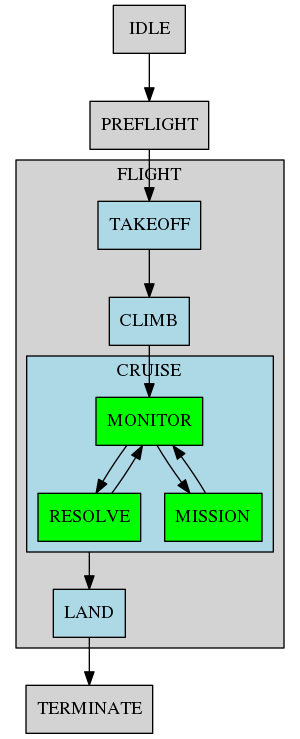
* command: MAV\_CMD\_SPATIAL\_USER\_1
* param1: Traffic id
* param2: velocity - north component [m/s]
* param3: velocity - east component [m/s]
* param4: velocity - up component [m/s]
* param5: lat [degrees]
* param6: lon [degrees]
* param7: alt [m]

**Launching ICAROUS**

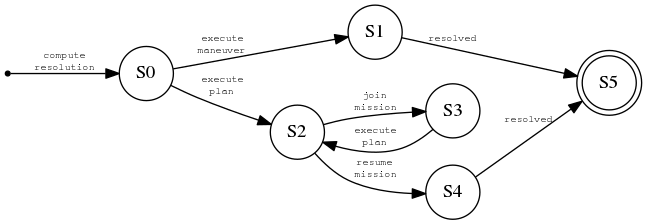
* Create an Icarous class. The class constructor takes a string of command line parameters and a class implementing the Mission interface as inputs.
* Launch icarous using the run() method.
* ICAROUS expects the following command line arguments:
  + -v (flag enable warning/error message outputs to console).
  + --sitl sitlhostaddress inputport (Not used if using --px4)
  + --com comhostaddress inputport outputport
  + --mode [active/passive/passthrough]
  + --px4 serialport baudrate (Not used if using --sitl)
* Send the MAVLink command message MAV\_CMD\_MISSION\_START with param1 set to 0 after the flight plan and required geofences have been input. ICAROUS will start monitoring the operations from takeoff to landing.

**Finite State Machines**

Conflict detection and resolution is established using hierarchically structured finite state machines. The hierarchy is show in the following figure:

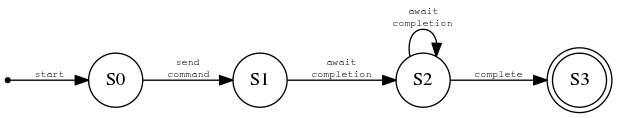


The above phase based state transition is implemented by the Aircraft class. The execution of a resolution is governed by the following state machine:



The initial state [S0] computes a resolution based on the available conflicts in the conflict queue. Some conflicts are resolved using a resolution maneuver [S1] (e.g. avoid traffic by a sequence of guided maneuvers) whereas some conflicts are resolved using an elaborate resolution flight plan [S2] (e.g. plotting a path to avoid an obstacle). After a resolution plan, state S3 computes an intermediate plan to join the original mission. State S4 resumes the original mission once the vehicle is back on the original flight plan. State S5 is a final state free of conflicts.

Resolution flight plan is handled by the following state machine:



Resolution maneuvers are handled by the following state machine:

